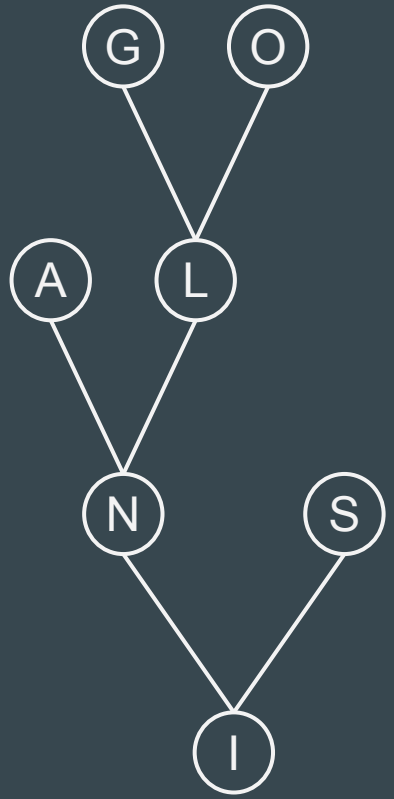
Trees

save trees plz they’re cool

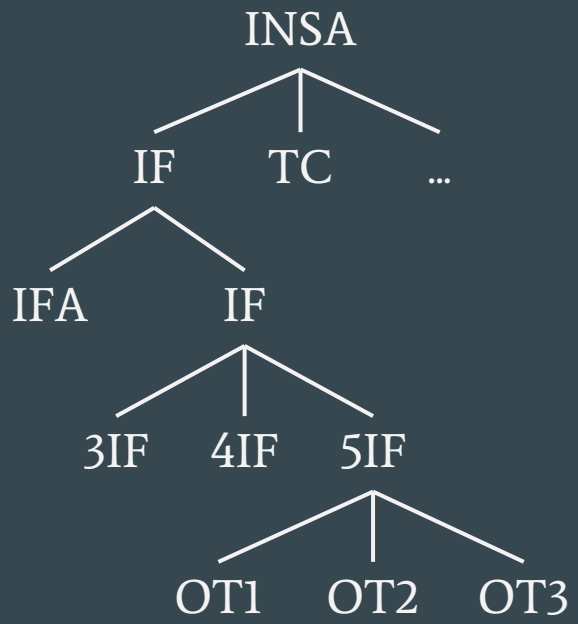


Introduction to trees

Last time we talked about multiple kinds of data structures:

- sequences and queues
- mathematical sets
- key-value associations

Now how to store the ***hierarchical structure*** of your data? (and many more cool things)



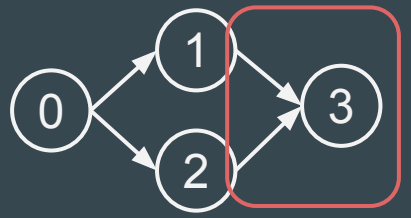
So what exactly is a tree?

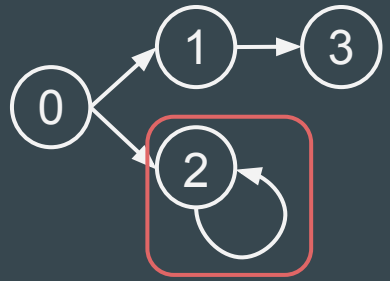
- recursively defined, starting from a *root* node
- each node has a value and a list of references to children nodes
- the root alone has no parent, all other nodes have *exactly one parent*

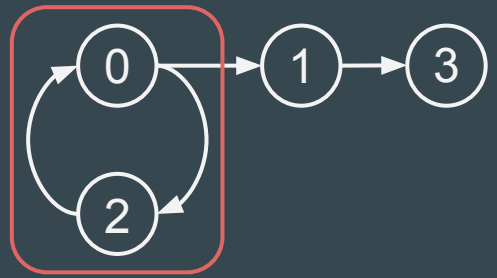
A tree cannot have *loops* !

(therefore, family trees are not real trees)

Not valid trees:

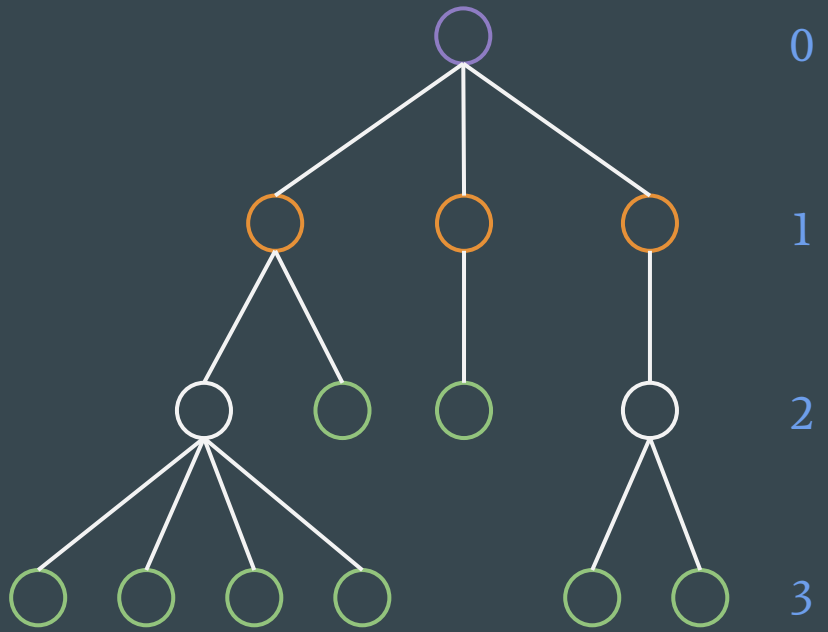




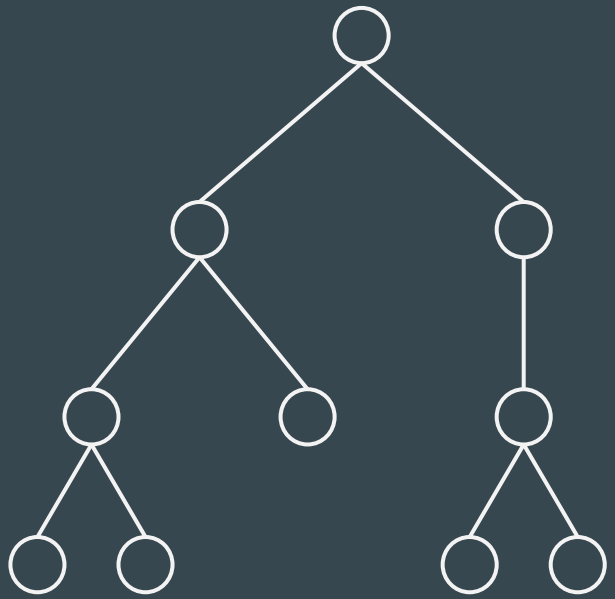


A few terms

- the root is the only node with no parent
- siblings are nodes which share a same parent
- the leaves are the nodes with no children
- the level or depth of a node is its distance to the root



Binary trees



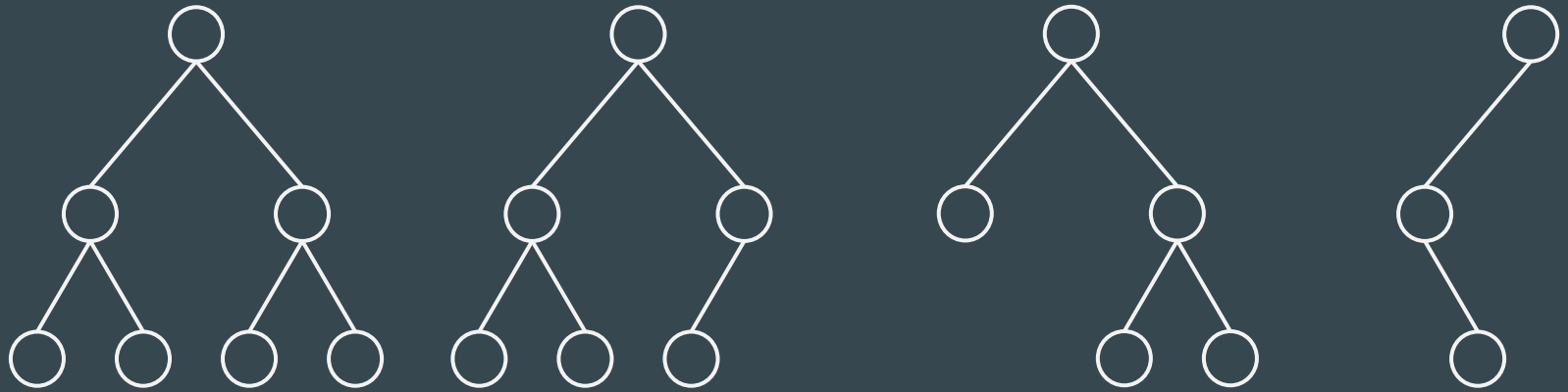
Today we'll talk mostly about *binary trees*.

That is, trees which nodes can only have *up to 2 children*.

C/C++	`template <typename ValType>`
	`struct BinaryTree {`
	`  ValType value;`
	`  struct BinaryTree<ValType>* left;`
	`  struct BinaryTree<ValType>* right;`
	`};`

Python	`BinaryTree = namedtuple('BinaryTree',` `  ['value', 'left', 'right'])`

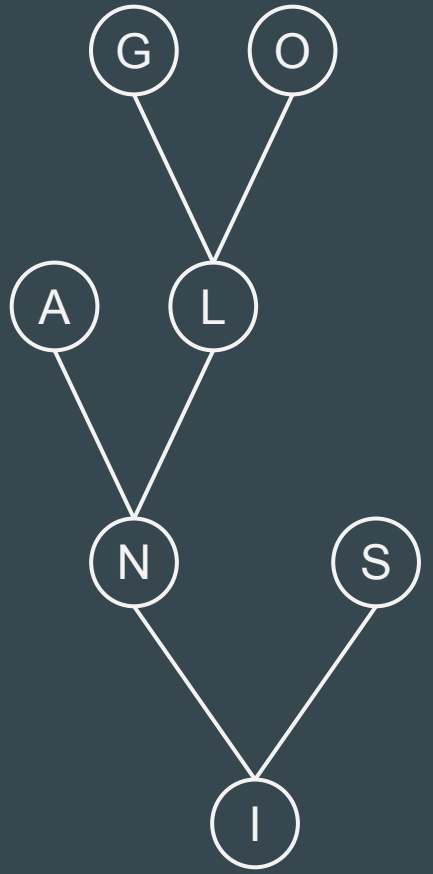
Binary trees: special cases



perfect	complete	full	degenerate
all levels are full	all levels are full except the last one where leaves are accumulated on the left	all the nodes have either 0 or 2 children	all the nodes have 0 or 1 children (linked list)

Binary tree traversal

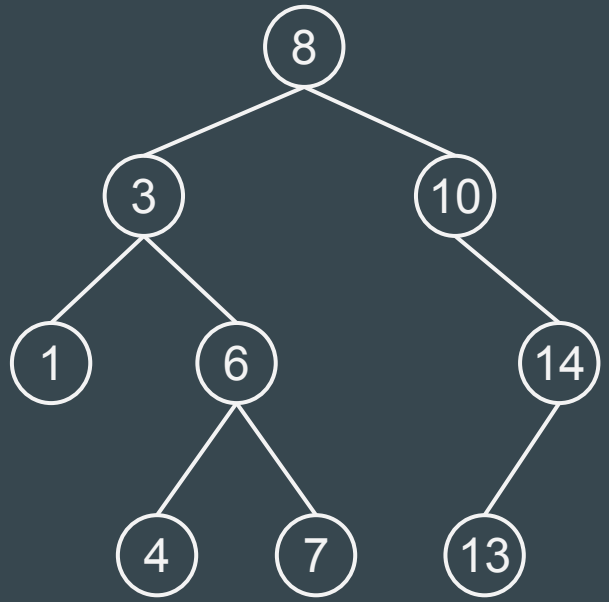
Pre-order	(préfixe)	I N A L G O S
In-order	(infixe)	A N G L O I S
Post-order	(postfixe)	A G O L N S I
Breadth-first	(en largeur)	I N S A L G O



Pre-order, in-order and post-order are all variants of depth-first traversal.

The difference is if the value is read, before, after or in-between the traversal of the left and right subtrees.

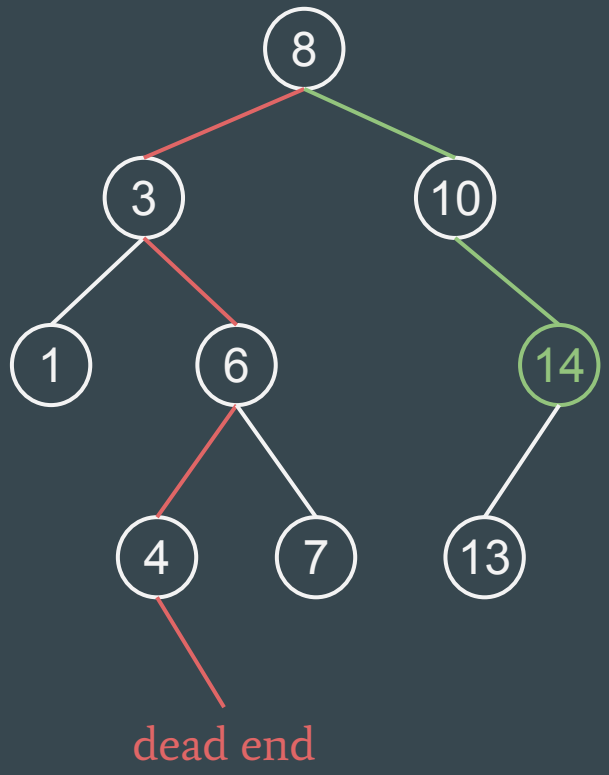
Binary search trees - definition



Goal: keeping a collection of values sorted while minimizing the cost of insertion and deletion

- it's a binary tree
- each node is:
  - >= every node in its left subtree
  - <= every node in its right subtree

Binary search trees - searching



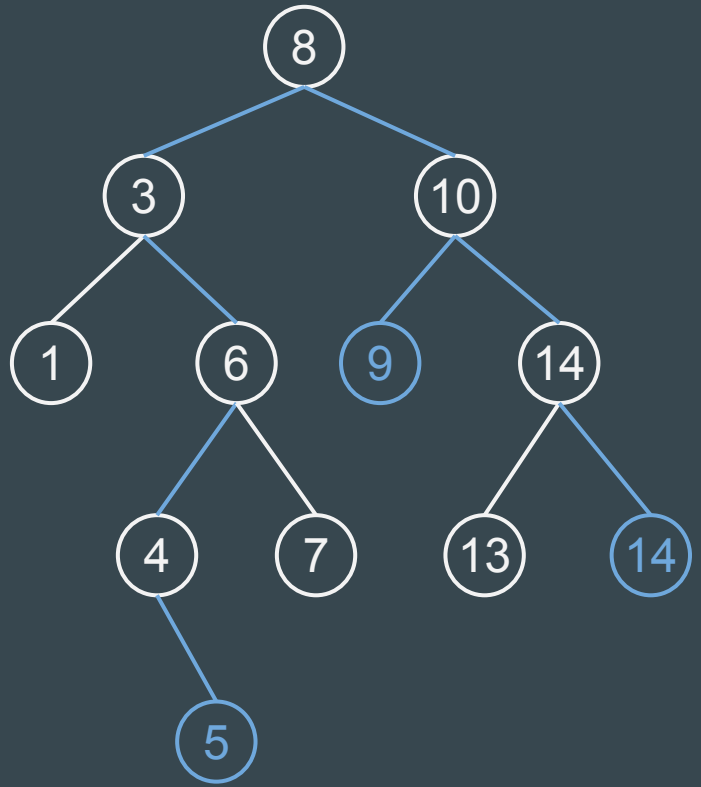
We want to find a specific value in the tree

➔ we just need to compare it with nodes starting from the root, and turning left or right depending on the result of the comparison

➔ if we encounter a dead end, the value isn't in the tree

In this tree, 14 is found, 5 isn't

Binary search trees - inserting



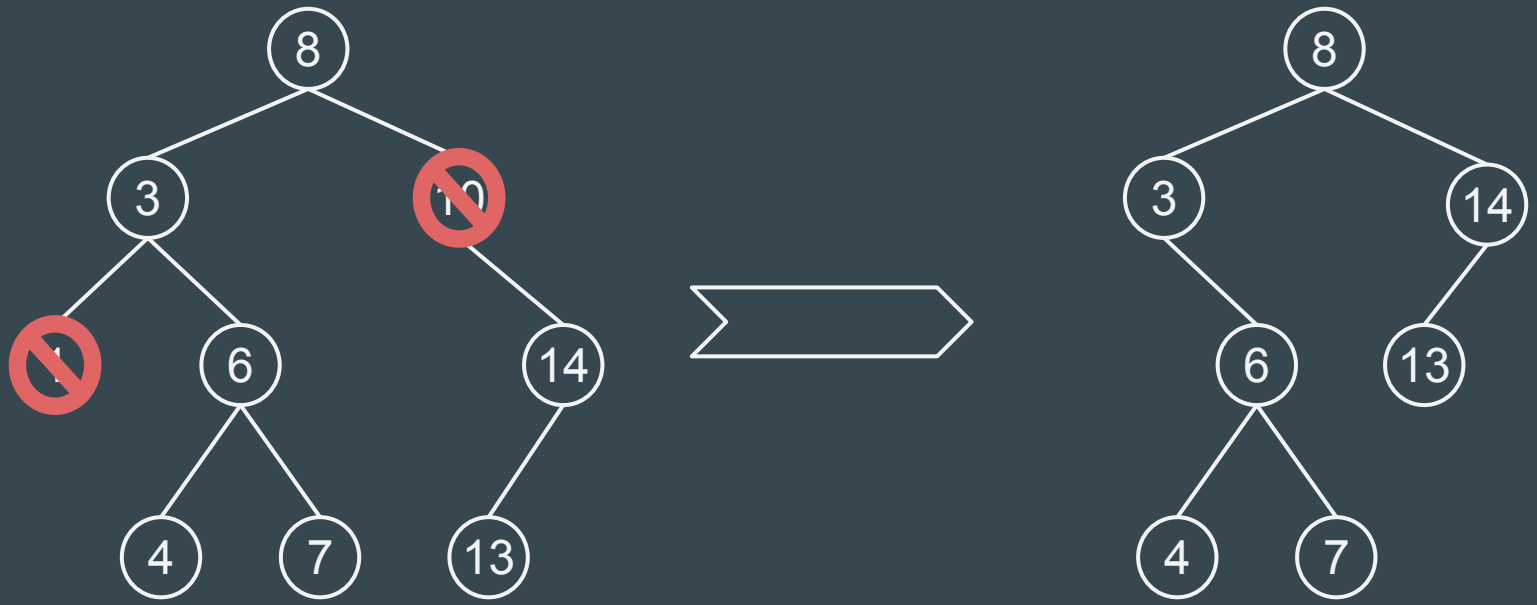
Insertion is very similar to search but:

- if the value is equal, you keep going down
- when you reach a dead end, add the value

Let's insert 5, 9 and 14

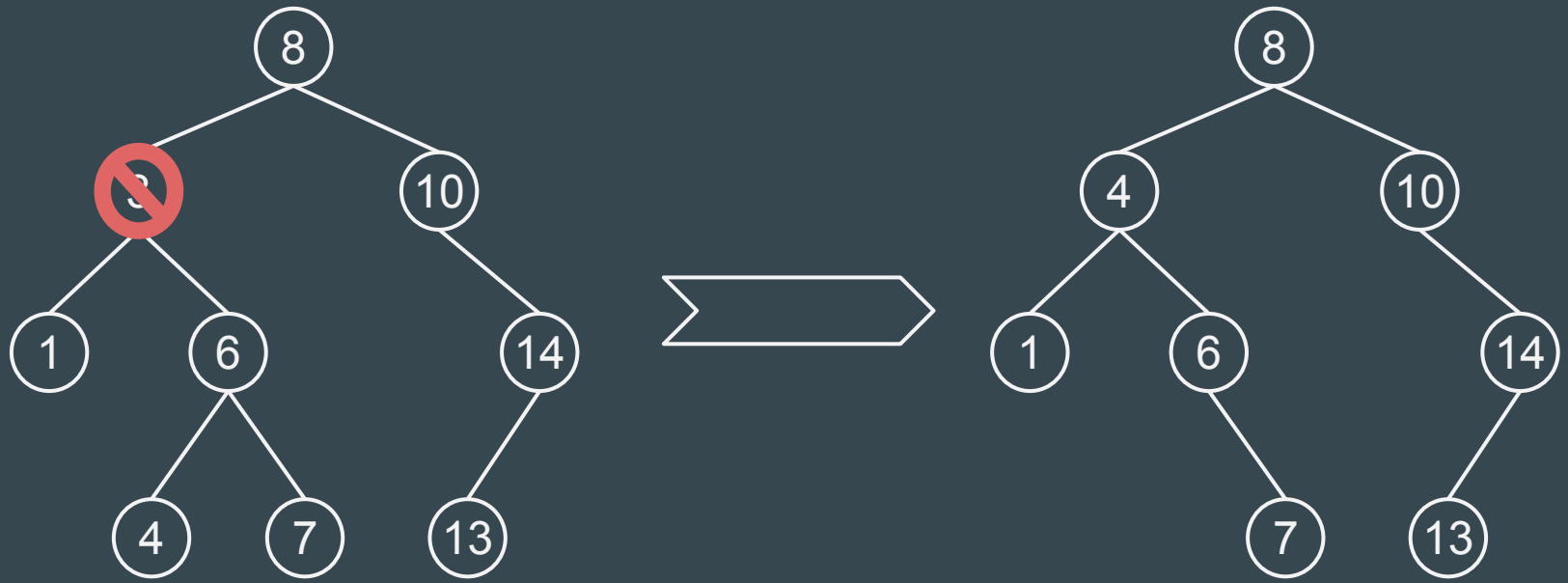
(in this BST we accept to have a same value multiple times, we then choose arbitrarily whether to insert it on the left or on the right)

Binary search trees - deleting



no children: simply delete the node / 1 child: replace the node by its child

Binary search trees - deleting



2 children: give the node the value of the rightmost node in the left subtree or the leftmost in the right subtree, and delete this other node (which has 0 or 1 child)

Binary search trees - Python implementation

I made a full implementation on GitHub (`trainings-2018`, W11, `binary_search.py`)

I encourage you to **read it** and **try it**

```
bst = BinarySearchTree.from_list(
    [8, 3, 10, 1, 6, 4, 7, 14, 13])
bst.insert(9)
bst.delete(3)
bst.draw()
```

```
      8
  1      10
    6   9    14
   4 7      13
```

```
bst.traversal()
```

```
[1, 4, 6, 7, 8, 9, 10, 13, 14]
```

Binary search trees - degeneration

The ideal case would be a balanced tree → the depth is $O(\log_2 n)$
But the tree can be very unbalanced → the depth is $O(n)$

Usually, after a lot of insertions and deletions trees are more and more unbalanced

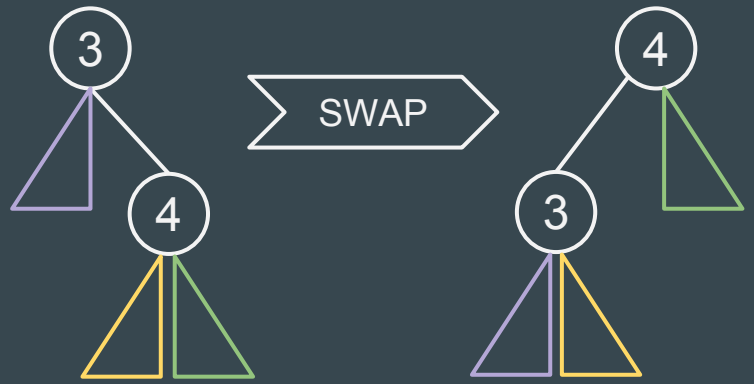
➔ try: `bst = BinarySearchTree.from_list([random.randint(0,50) for _ in range(50)])`

The cost of searching, inserting and deleting elements is proportional to the depth of the tree. We want to keep this depth as small as possible

Binary search trees - balancing

To avoid degenerating, some advanced variants use self-balancing

→ they reorganize their internal structure when the tree becomes too unbalanced

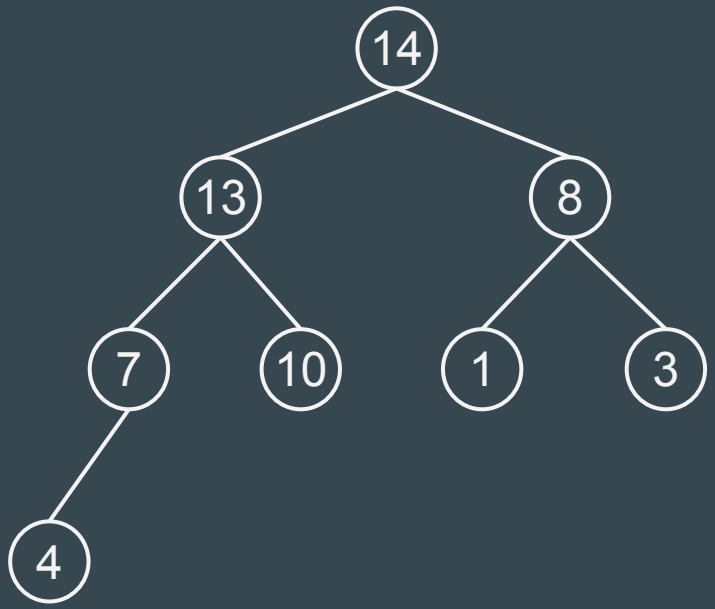


AVL trees: https://en.wikipedia.org/wiki/AVL_tree → see HackerRank exercise

Red-black trees: https://en.wikipedia.org/wiki/Red-black_tree

Periodical balancing: https://en.wikipedia.org/wiki/Day-Stout-Warren_algorithm

Heaps - definition

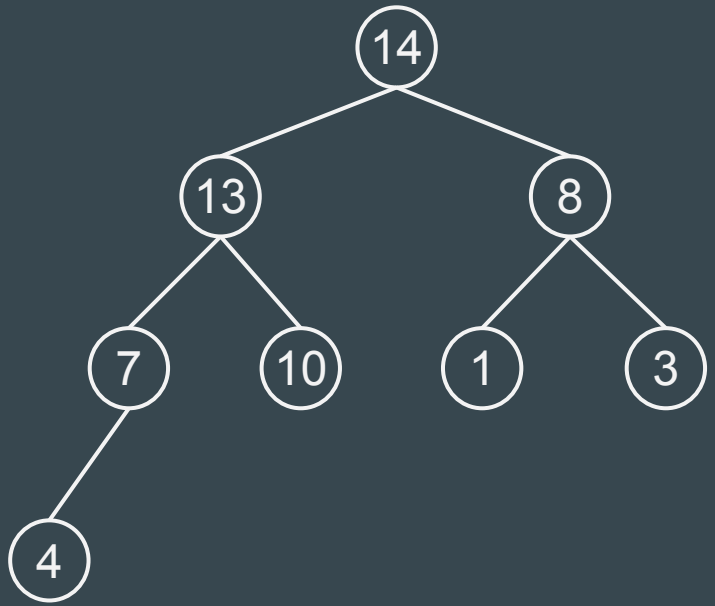


Goal: always retrieving the biggest element of a collection efficiently → priority queues

(we'll use max-heaps but min-heaps are similar)

- it's a ***complete*** binary tree
- each node is bigger than all elements in its childrens' subtrees

Heaps - memory representation



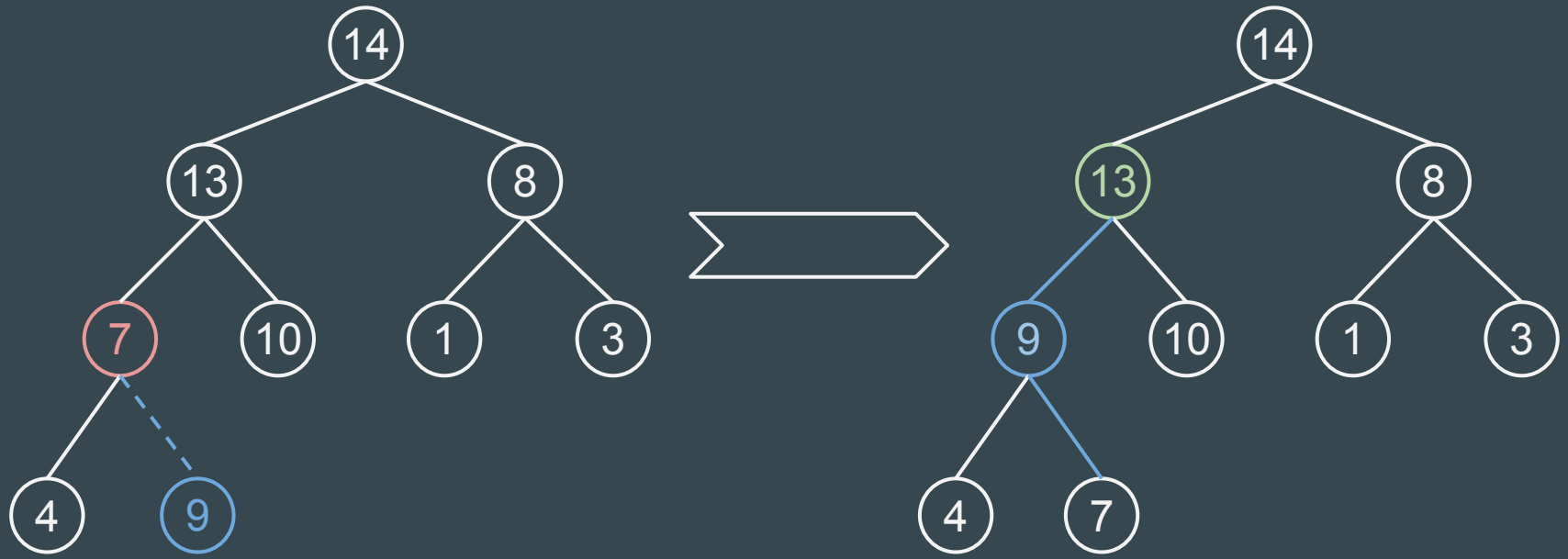
The binary tree is *complete*

Therefore it can be stored in an array:

`[14, 13, 8, 7, 10, 1, 3, 4]`

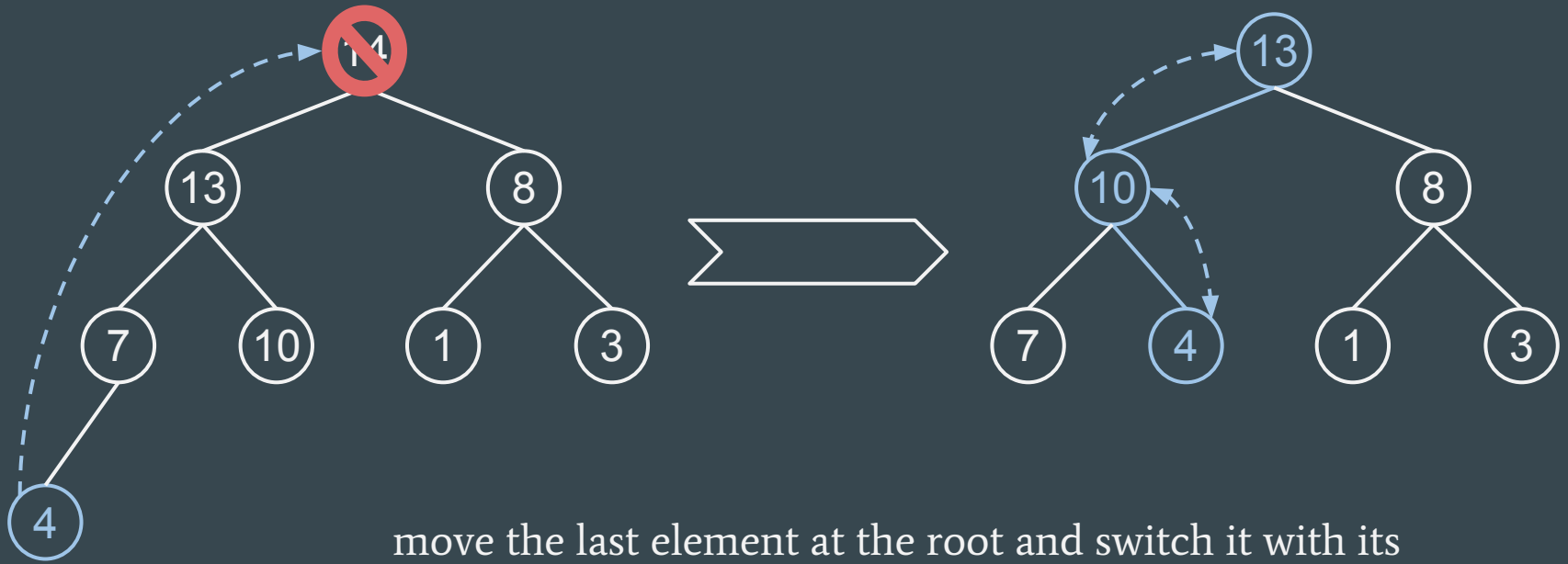
- the `i`th node's parent is `⌊(i-1)/2⌋`
- the `i`th node's children are i*2+1 and i*2+2

Heaps - inserting an element



insert the element at the end and switch it with its parent until it is bigger

Heaps - extracting the root (biggest element)



move the last element at the root and switch it with its biggest child until it's bigger than both children

Heaps - complexity

Heaps are ***always complete*** binary trees

➔ so their depth is $O(\log_2 n)$

Hence the following complexities:

	average	worst
find biggest	`O(1)`	`O(1)`
remove biggest	`O(log n)`	`O(log n)`
insert	`O(1)*`	`O(log n)`

* not trivial - explanation here

In Python, module `heapq` in the standard library

In C++, `make_heap`, `pop_heap` and `push_heap` in `<algorithm>`

Heapsort

Heapsort a beautiful sorting algorithm that can be performed in place, with worst-case complexity O(n log n) - though on average slower than a well-implemented quick sort

- elements of the array are inserted in a heap
- the array is then filled by taking iteratively the biggest element in the heap

To perform it in place, one part of the array is the heap and the other contains the remaining elements

What are the other uses of trees?

Trees are used everywhere, it would require hours to have a fair overview of the topic

- databases use B-trees
- text editors use ropes
- spell checkers and autocomplete use tries
- 3D engines use octrees
- IP routing uses radix trees
- bioinformatics and data compression use suffix trees
- peer-to-peer systems, version control systems, NoSQL databases use Merkle trees
- compilers use spaghetti stacks
- map services use R-trees

etc...

Conclusion

That’s enough for now :)

Slides by Louis Sugy

Thanks Wikipedia for being so great <3